KBG.EXE

A Fuzzy Logic Knowledge Base Generator for the MC68HC11 and MC68HC05 Inference Engines

Program KBG.EXE creates and edits a knowledge base, runs a software simulation of an inference engine, and uses the simulation engine to create a 2-D plot of the control surface described by your knowledge base. This tutorial demonstrates the use of program KBG.EXE to create a new knowledge base file for a FUZZY logic application. KBG.EXE acts as a preprocessor to generate a knowledge base for either the MC68HC11 or MC68HC05 inference engine, FUZZY11B.ASM or FUZZY05B.ASM, using natural language inputs. At present, only FUZZY11B.ASM is available; FUZZY05B.ASM will be available soon.) The inference engines require the FUZZY knowledge base to be in the form of assembly language constants (FCB - Form Constant Byte directives) scaled from 0 to 255. This is a data structure that the MC68HC11 or MC68HC05 assemblers can understand. KBG.EXE performs the following operations:

1. Creates a knowledge base in a natural language format.
2. Loads an existing knowledge base.
3. Saves the current knowledge base.
4. Edits a knowledge base.
5. Generates a print file.
6. Generates the MC68HC11 or MC68HC05 assembly code file.
7. Runs a software emulation of the inference engine.
8. Displays a 2-D slice of the control surface.

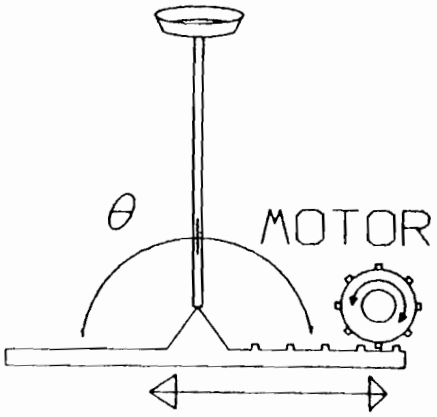
The current version of KBG.EXE is compiled to accept:

1. 8 Inputs.
2. 4 Outputs.
3. 8 Membership functions per input.
4. 8 Membership functions per output.
5. 1024 rules.
6. Any number of inferences per rule.
7. 15 characters per name (input, output, member).
8. 4 points per input member.
9. 1 point per output member.

Future versions of this program will be posted on Motorola's Freeware BBS as they become available. You can reach the Freeware BBS at (512) 891-FREE. Connect at 2400, 1200, or 300 baud with 8 bits, No Parity, and 1 Stop Bit. The Freeware BBS System also has the code for the MC68HC11 and MC68HC05 Inference Engines.

This tutorial uses the IBM-PC version of KBG.EXE to demonstrate creation of a new knowledge base. KBG.EXE requires a VGA graphics adapter. If available, use a '386 or '486 based machine; some functions may be unacceptably slow on a PC equipped with an older microprocessor.

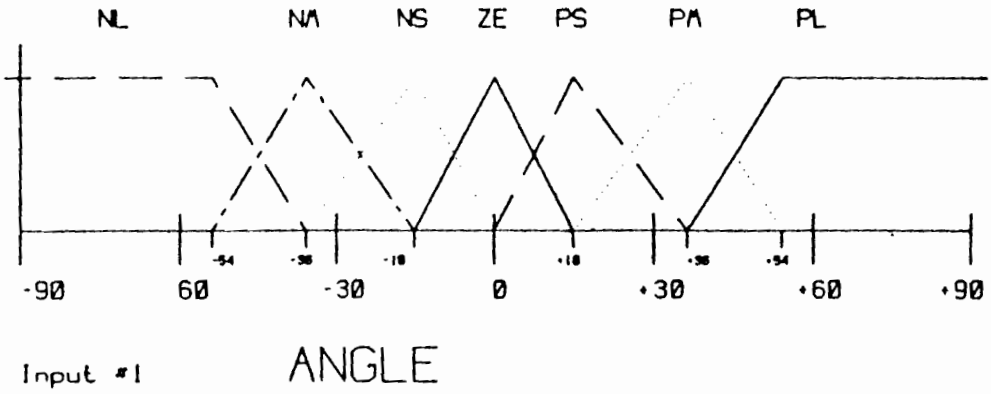
The FUZZY Logic example presented here shows the coding of the classic inverted pendulum balancing problem. This problem consists of variable amount of mass atop a variable length rod. The base of the rod is moved to balance the rod and mass. This example shows how to code the problem as presented in chapter 8 of Bart Kosko's book, *Neural Networks and Fuzzy Systems* (Prentice Hall).

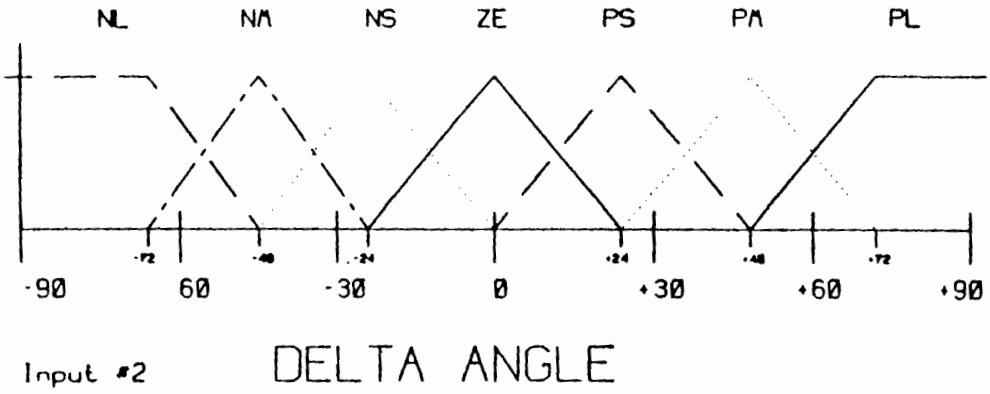


The figure to the left shows a schematic of the problem situation. For this example, consider only motion on the X-Y axis, not the Z axis. This simplifies the problem to only two inputs. Input 1 is the angle (phi) of the pendulum to the vertical. Input 2 is the rate of change of the angle (delta phi) in degrees per second.

Once you identify the inputs, you must create membership functions for each input. Input 1 requires seven membership functions. A negative deflection of the angle is within one of three members. A large deflection, -36 to -90 degrees, is labeled **Negative_Large** (**NL**). A lesser defection, between -18 and -54 degrees, is labeled **Negative_Medium** (**NM**). A deflection of 0 to -36 degrees is labeled **Negative_Small** (**NS**). The three positive deflections are symmetrical, and are similarly defined. A deflection of **Zero** (**ZE**) occurs from -18 to +18 degrees. As is the case with fuzzy logic, a particular deflection can fall into more than one membership function. For example, a deflection of -5 degrees is **Zero** to a great extent and **Negative_Small** to a lesser extent.

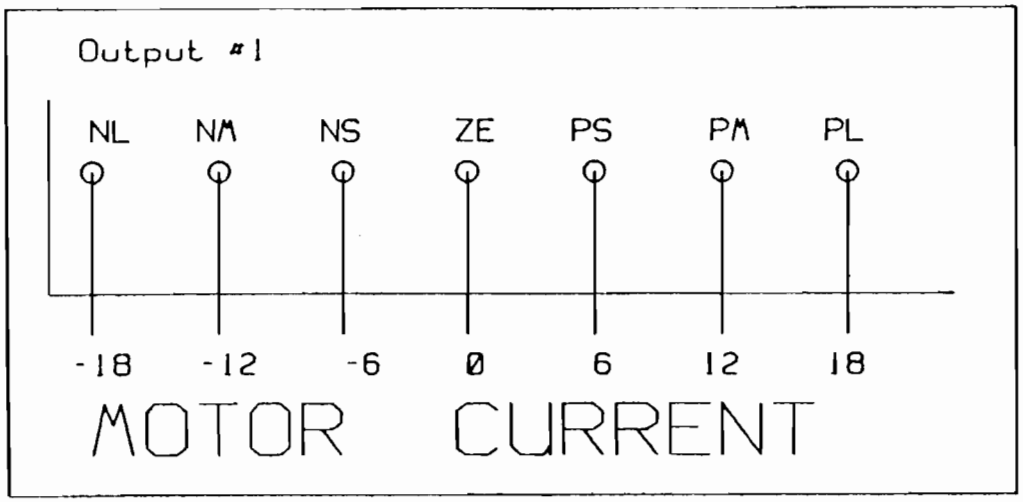
Input 2 is almost identical, but refers to rate of change. Thus, a value of +30 means +30 degrees / second change. See the following diagram for the details of input membership functions.





This system has only one output. The output controls the motor rotation to move the balance point right or left. The membership functions for the output are shown in the following figure. Three levels of drive current for the motor in each direction are defined, and zero current is defined for the stopped condition. The seven motor current levels are:

NL	**Negative_Large**
NM	**Negative_Medium**
NS	**Negative_Small**
ZE	**Zero**
PS	**Positive_Small**
PM	**Positive_Medium**
PL	**Positive_Large**



The MC68HC11 and MC68HC05 Inference Engines define outputs as singletons; thus, only one discrete point defines each output.

Now to begin the coding exercise; the first step is to execute KBG.EXE.

C:> kbg

As KBG begins, you see a signon screen with the Motorola copyright notice. Press any key to proceed to the Main Menu.

```
Knowledge Base Generator (KBG) V2.10
(Copyright Motorola Inc. 1991,1992)

          KBG Main Menu.
L - Load Knowledge Base File.
S - Save knowledge Base File.
E - Edit Knowledge Base File.
P - Create PRINT Output File.
A - Make  Assembly Code File.
D - Draw Cntr'l Surface Plot.
X - Show  Engine   Execution.
Q - Return to DOS from KBG.
```

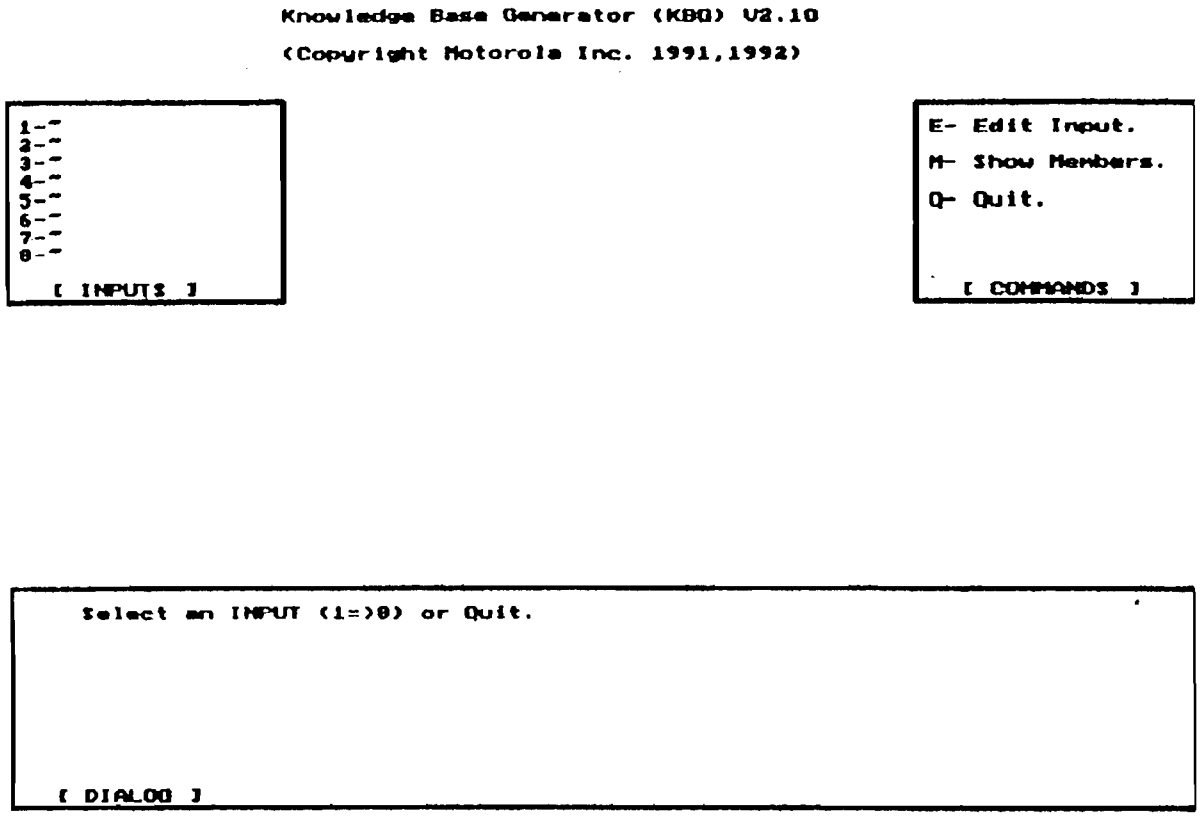
KBG.EXE has an on-line help system that you can activate by pressing the H key. The help system displays help menus and/or text in a window. You can use the up and down arrow keys to select a topic for help. The ENTER key activates the help text or submenus for the selected topic. Use the ESCAPE key to exit the help system and the BACKSPACE key to back up to the previous display, either text or menu.

At this point, KBG.EXE contains no data to describe a knowledge base. With KBG.EXE, you can load a previously created knowledge base or create a new one. This tutorial demonstrates creating a new one. Begin by selecting E from the Main Menu. This clears the screen and calls up the Edit Menu, displayed by the subsystem you use to describe a new base or to modify an existing base.

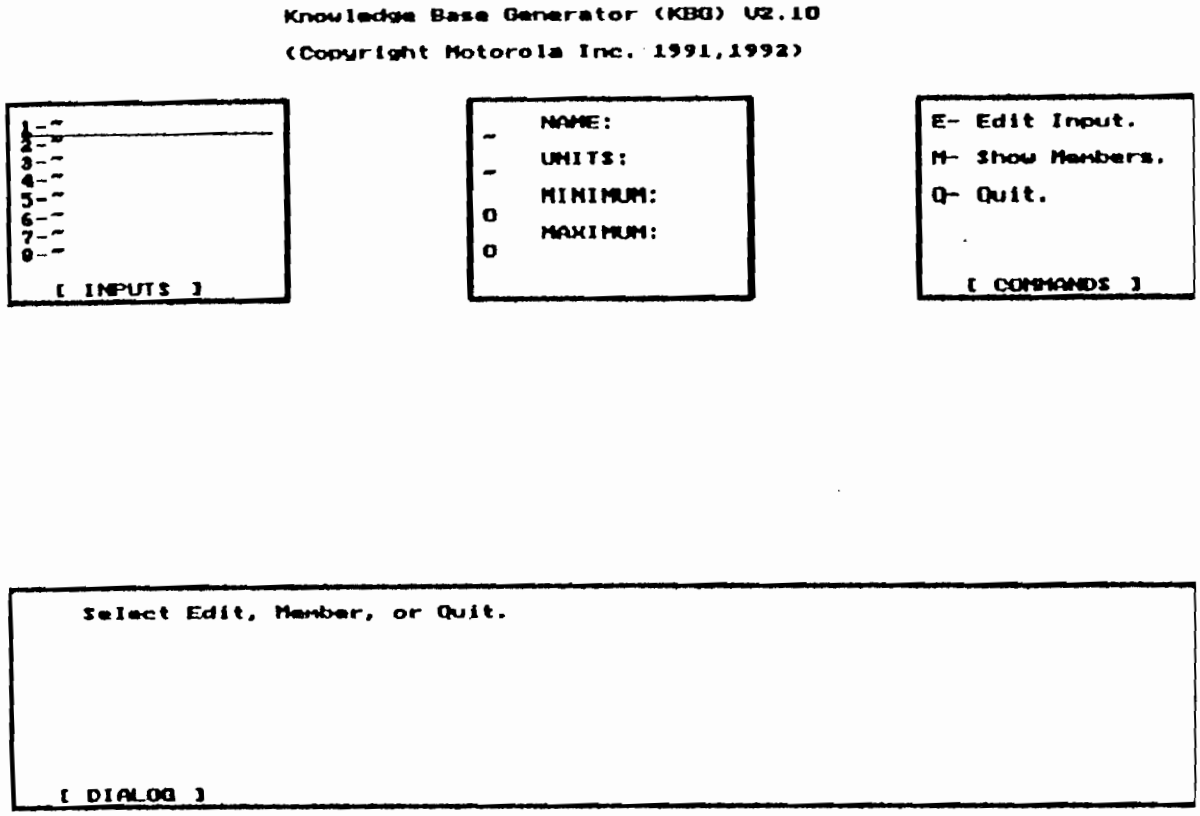
```
Knowledge Base Generator (KBG) V2.10
(Copyright Motorola Inc. 1991,1992)

          KBG Edit Menu.
I - Inputs:   Add, Edit, Delete, Members.
O - Outputs:  Add, Edit, Delete, Members.
R - Rules:    Add, Delete Rules.

Q - Quit to MAIN Menu.
```

You may add either the inputs or the output now, in any order. However, it would make no sense to define rules now, with no inputs, outputs, or membership functions to reference.

In this case, add the inputs first. Selecting I for inputs brings up the data entry screen for inputs, as shown in the following figure. The eight inputs are displayed in a window at the upper left corner of the monitor screen. Each undefined input shows a single tilde (~) as the input name. In this case, all inputs are undefined and all inputs show tildes as their names.

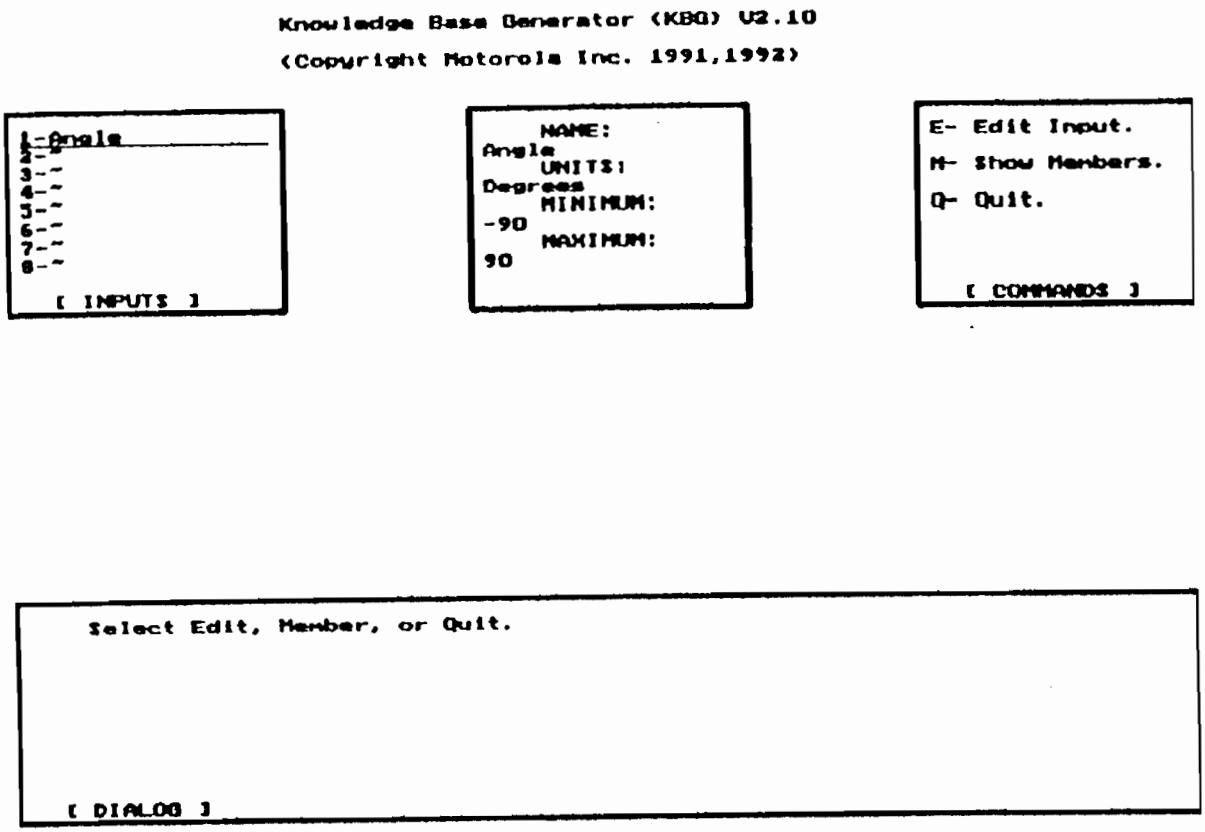


Next, press a number key, **1** through **8**. This selects an input for editing. The inputs can be defined in any order. The selected input is underlined in the left window. Another window is opened in the upper center of the monitor screen, as shown in the following figure. This window shows the current Name, Units, Minimum, and Maximum for the selected input. Press the **1** key and note that the Name and Units for input #1 are undefined. Both the Minimum and Maximum are zero (undefined).



Next, press the **E** key to edit input #1. The NAME field in the center window is underlined to show that it is selected. Pressing the ENTER key selects the NAME field for edit and opens an edit window below the other two. At the prompt (>) in the edit window, type the name of input #1 - **Angle**. Press the ENTER key; the edit window should vanish and the program should display the word Angle in the NAME field in the center window. This shows that the name you typed has been placed in the knowledge base. The underline is moved to the UNITS field to show that it is now selected instead of the NAME field. Press the ENTER key and enter the word **Degrees** as the name for UNITS. In the same way, enter **-90** for MINIMUM and **90** for MAXIMUM.

After you enter the last item for input #1, the program removes the center window and writes the name Angle in the left window as the name for input #1. The other data for this input is stored in the knowledge base but is not displayed on this screen.



Now, select input #2 for edit and enter **Delta_Angle** for NAME, **Degrees_Second** for UNITS, **-90** for MINIMUM, and **90** for MAXIMUM. Both inputs are now defined, but contain no membership functions.

To enter membership functions, select Input #1, Angle, and press the M key for Members. The program clears the screen and displays three windows, as shown in the following figure. The top window shows membership functions on a graph; the X-axis represents values from MINIMUM to MAXIMUM and the Y-axis represents degree of membership. Each point on the X-axis is represented by a two number pair, such as -90,0 or 0,128. The first number of the pair represents the natural language input value and the second number is the internal representation. The second is a number from 0 to 255 (one byte). In this example, the value -90 is the same as MINIMUM and is represented as zero. Likewise, the input value +90 is the same as MAXIMUM and is represented as 255. All input values between -90 and +90 are scaled to a number 0 through 255. (You do not have to know the internal representation, but knowing may help you to understand the final assembly language output).

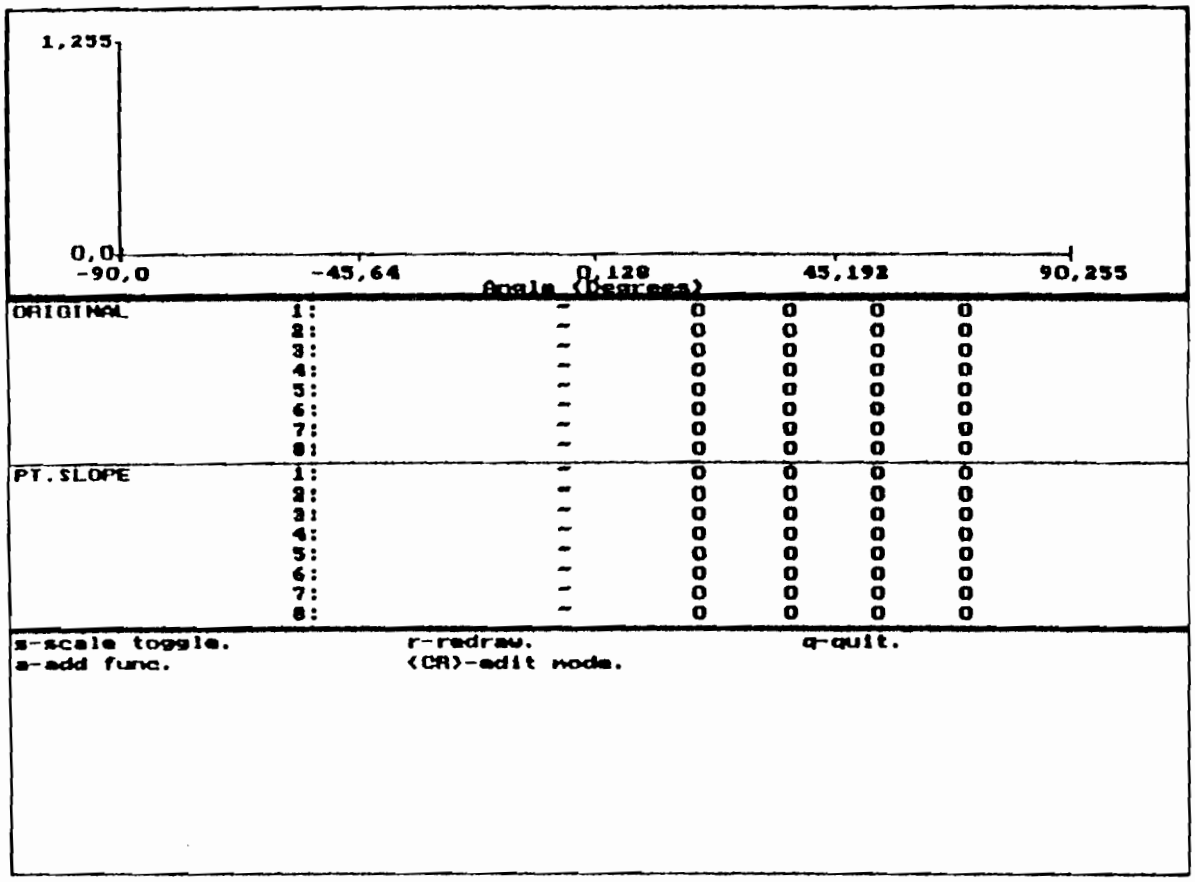
The numbers on the Y-axis are represented by the same type of two-number pairs. The first number, 0 through 1 (0% through 100%), is the degree of membership and the second number is the internal representation.

The center window shows the names and values of membership functions in a tabular format using two tables. The ORIGINAL table shows members in input

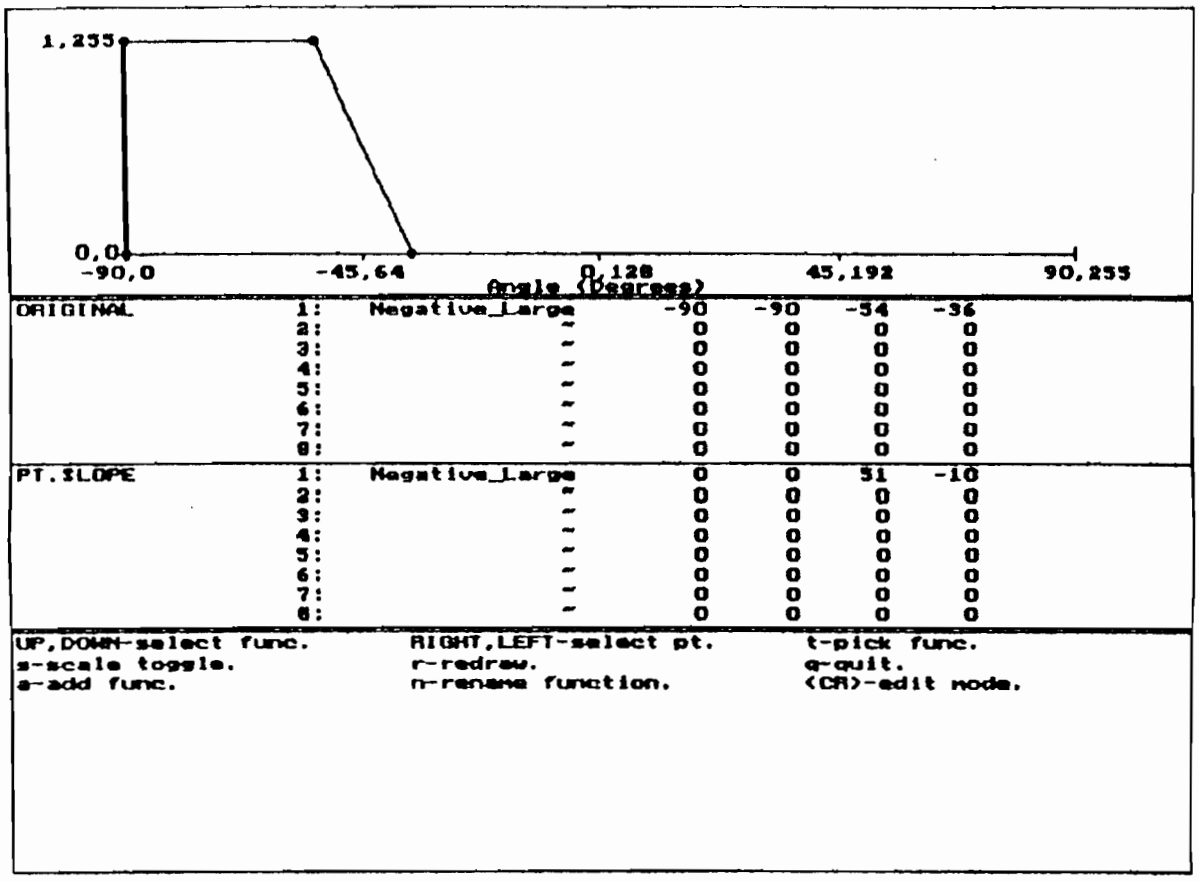
format. The table labeled PT.SLOPE shows the same data represented as point-slope pairs.

The bottom window displays available commands. It changes as data entry modes change. The initial key commands are s, r, q, a, and <CR>, listed below:

s	exchanges the position of the two tables in the center window. No data is modified.
r	redraws the entire screen; no data is modified.
q	quits, and returns to the preceding menu.
a	adds a new membership function.
<CR>	activates the edit mode.



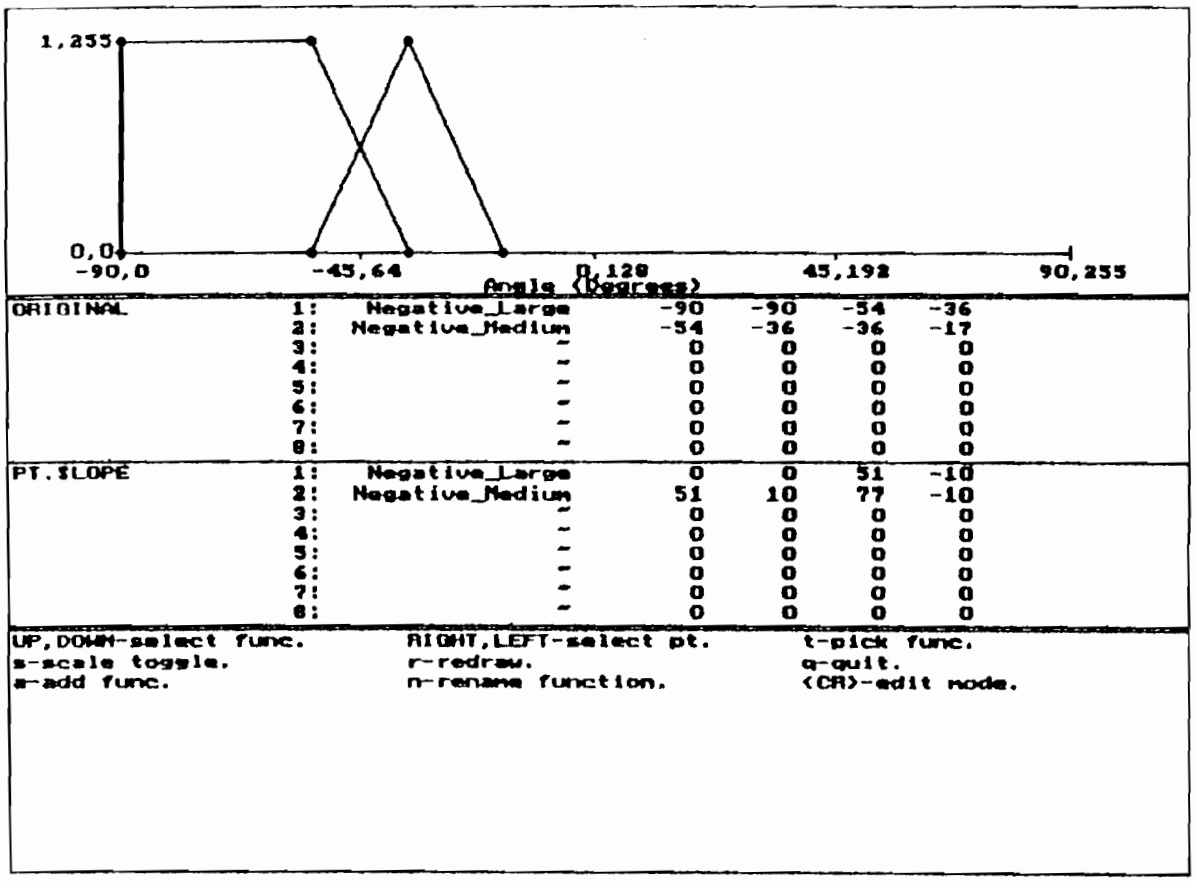
Continuing this example, press the **a** key to begin entry of the first membership function. Text in the bottom window now requests the name of the membership function. Type **Negative_Large** as the name for the first membership function. Names can be as long as 15 characters, with upper case, lower case, and/or underscore characters. The program displays the name in the two tables and requests point #1. Enter **-90**. The program writes the value -90 into the tables, draws a line on the graph at -90 on the X-axis, and requests point #2. Enter **-90**. The same scenario repeats. Enter **-54** for point #3 and **-36** for point #4.



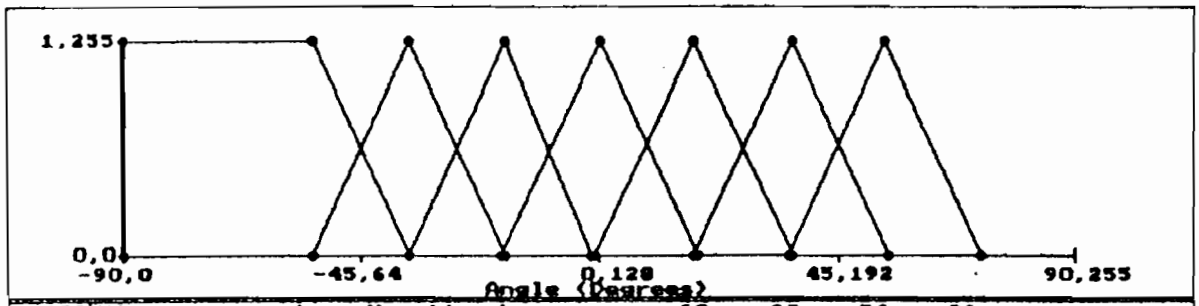
The first membership function has been entered. Note that this is a special case trapezoid in which the first two points are at the same point on the X-axis. All input membership functions are trapezoids (four points), but may have points at the same location. The command menu has changed, but just ignore that for a moment and enter the second membership function. The second member is called **Negative_Medium** and has points **-54, -36, -36, -18**. Press the a key and enter the member. Note that this also produces a special case of a trapezoid - a triangle. The next four membership functions all have the same shape; generate them by copying member #2.

Member #2 is currently highlighted in the following figure. If it were not, you would press the up-arrow or down-arrow keys until it became highlighted. Press

the **t** key to select the highlighted member, then press the ENTER key. The command menu now offers the c (copy) command. Press the **c** key and supply the name **Negative_Small**. A new member is drawn on top of the current member. We press the right-arrow key and this moves the new member to the right on the X-axis. We press the right-arrow key until the leftmost vertex is at -36 on the X-axis. You can read this value from the first entry of the ORIGINAL table. Press ENTER and the copy operation is complete.



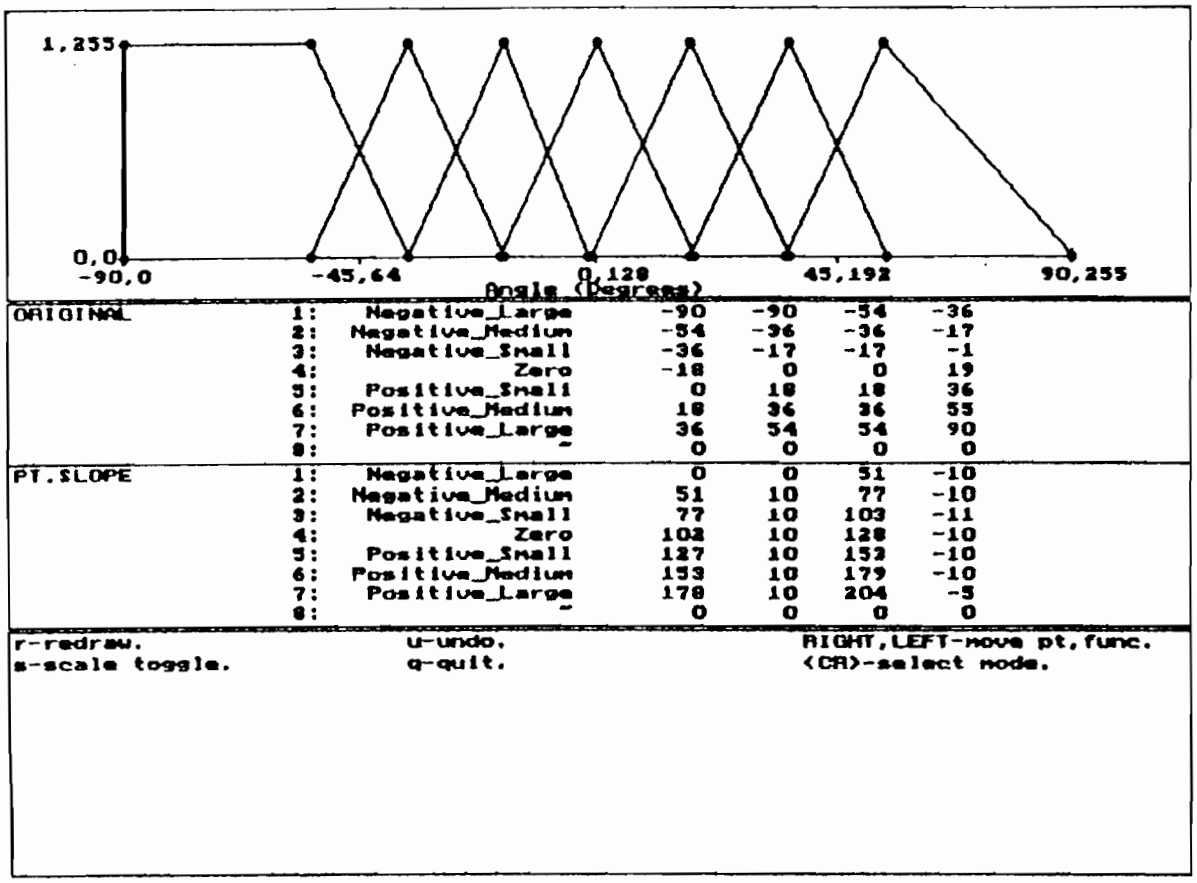
Repeat the copy operation for **Zero** (-18), **Positive_Small** (0), **Positive_Medium** (18), and **Positive_Large** (36). Now seven members are defined, as shown in the following figure, but the seventh member, **Positive_Large** has the wrong shape. You must edit it. Use the edit function to select any vertex of the member and move the vertex to another point on the X-axis.



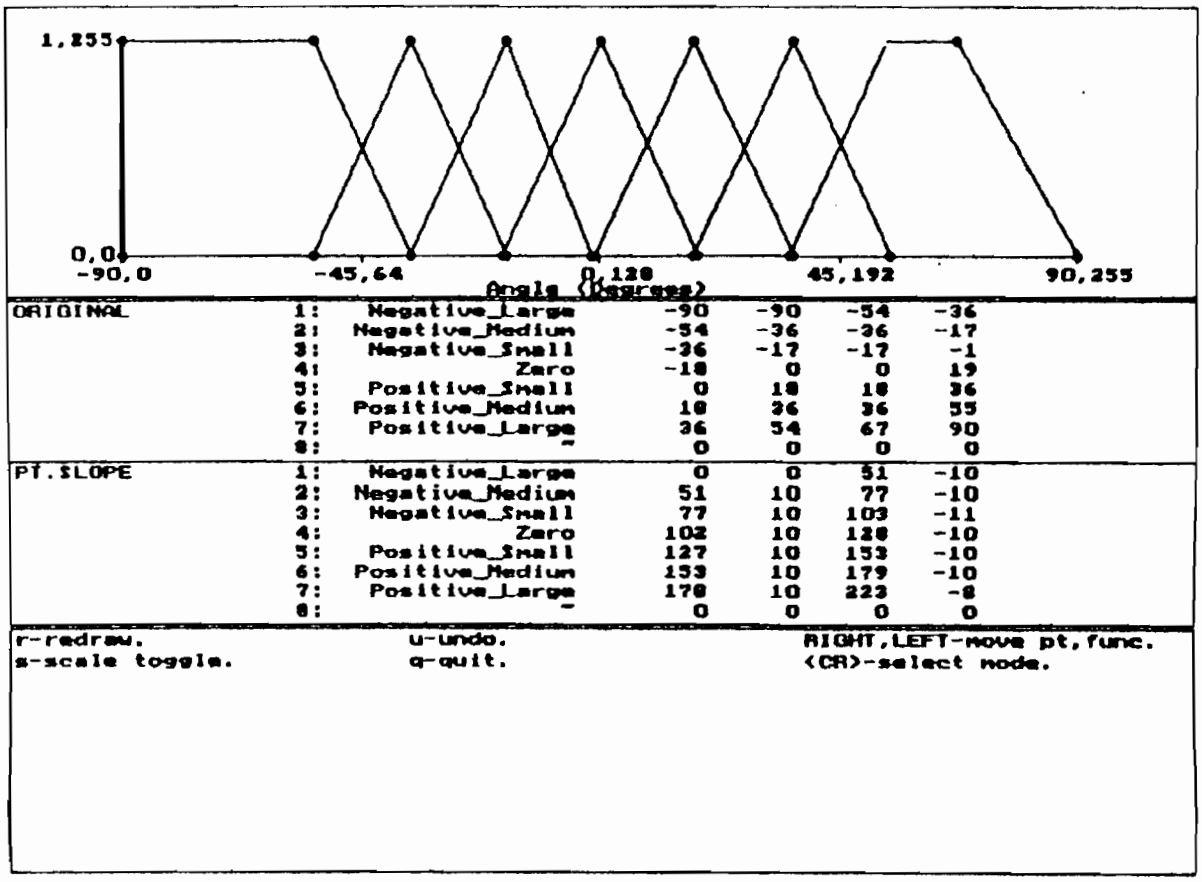
ORIGINAL	1:	Negative_Large	-90	-90	-54	-36
	2:	Negative_Medium	-54	-36	-36	-17
	3:	Negative_Small	-36	-17	-17	-1
	4:	Zero	-18	0	0	19
	5:	Positive_Small	0	18	18	36
	6:	Positive_Medium	18	36	36	55
	7:	Positive_Large	36	54	54	72
	8:		0	0	0	0
PT.SLOPE	1:	Negative_Large	0	0	51	-10
	2:	Negative_Medium	51	10	77	-10
	3:	Negative_Small	77	10	103	-11
	4:	Zero	102	10	128	-10
	5:	Positive_Small	127	10	153	-10
	6:	Positive_Medium	153	10	179	-10
	7:	Positive_Large	178	10	204	-10
	8:		0	0	0	0

r-redraw. u-undo. RIGHT,LEFT-move pt,func.
s-scale toggle. x-delete func. c-copy func.
q-quit. (CR)-select mode.

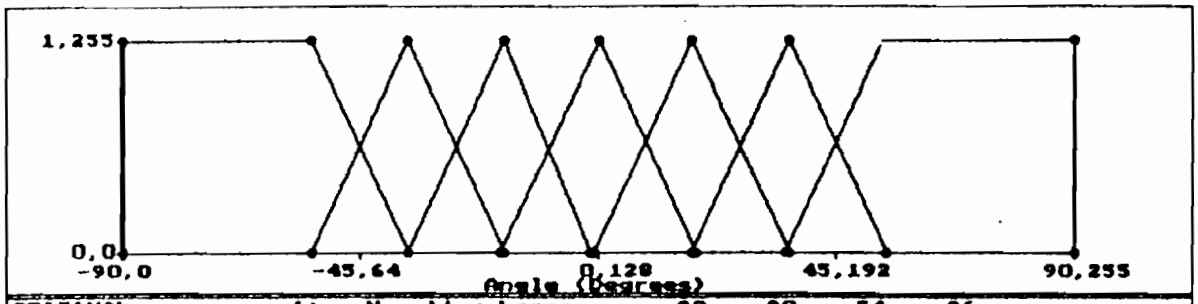
Use the up-arrow or down-arrow key until member #7 is highlighted and then press the **t** key to select the member. Press ENTER to enter the edit mode. Now the left-arrow and right-arrow keys can select one of the four vertices of the member. Select the right-most vertex, point #4, and press the ENTER key. Next, use the right-arrow key to move the vertex to the right until it reaches 90, as shown in the following figure.



Notice that the seventh member is no longer an isosceles triangle, but is now asymmetrical. The member needs more editing. Notice also that the values in the right columns of the ORIGINAL and PT. SLOPE tables have changed, consistent with the new location of vertex #4. Next, select vertex #3 and move it to 90. The following figure shows the screen with vertex #3 at 67, an intermediate point.



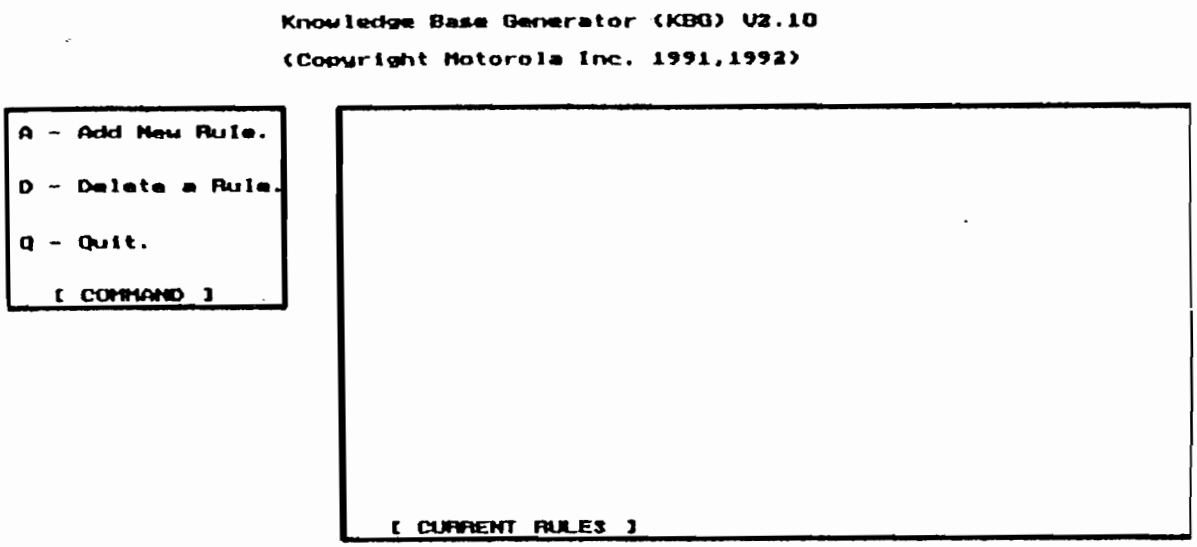
Normally, you would not stop at this intermediate point. This figure is shown to help you to understand the transition of the member from a triangle to a trapezoid. Again, notice the effect of moving the vertex on the values in the tables. When you continue to move the vertex to 90, the screen changes as shown in following figure. The membership entry is complete.



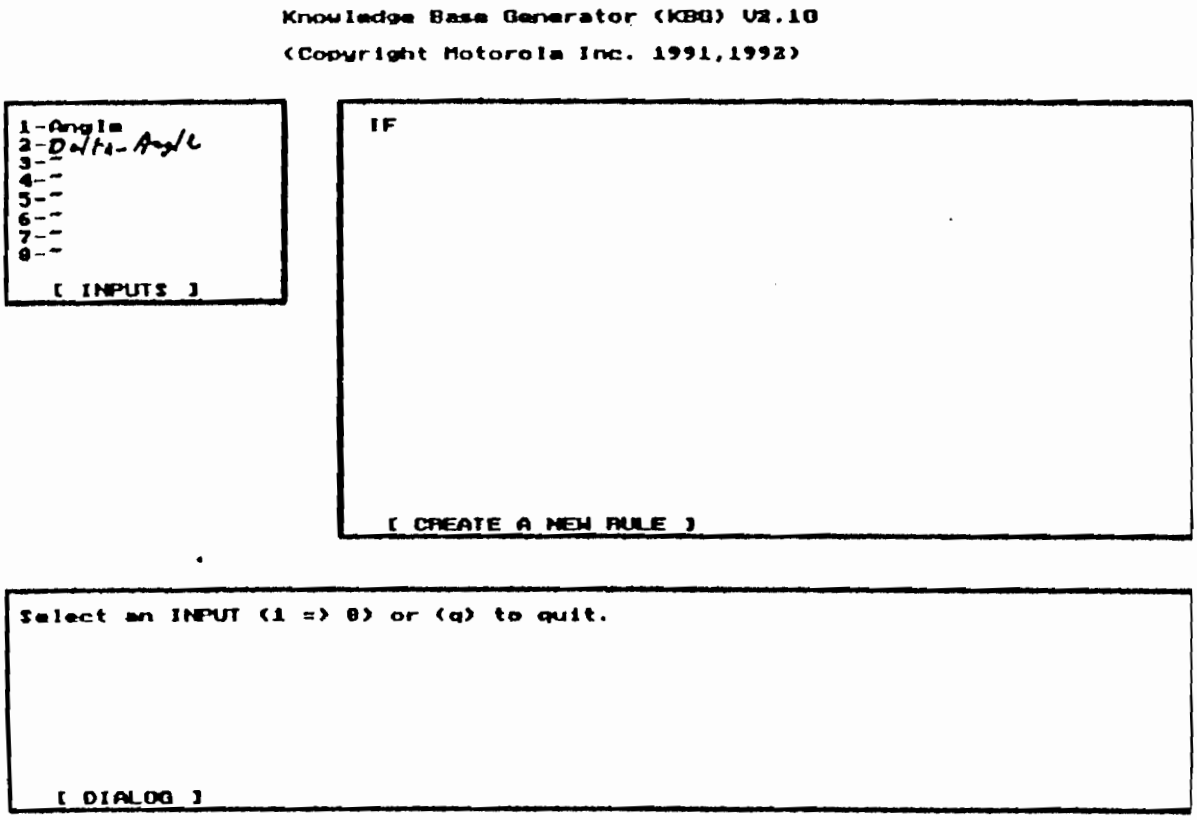
ORIGINAL	1:	Negative_Large	-90	-90	-54	-36
	2:	Negative_Medium	-54	-36	-36	-17
	3:	Negative_Small	-36	-17	-17	-1
	4:	Zero	-18	0	0	19
	5:	Positive_Small	0	18	18	36
	6:	Positive_Medium	18	36	36	55
	7:	Positive_Large	36	54	90	90
	8:	-	0	0	0	0
PT.SLOPE	1:	Negative_Large	0	0	51	-10
	2:	Negative_Medium	51	10	77	-10
	3:	Negative_Small	77	10	103	-11
	4:	Zero	102	10	128	-10
	5:	Positive_Small	127	10	153	-10
	6:	Positive_Medium	153	10	179	-10
	7:	Positive_Large	178	10	255	0
	8:	-	0	0	0	0

r-redraw. u-undo. RIGHT,LEFT-move pt,func.
s-scale toggle. q-quit. <CR>-select mode.

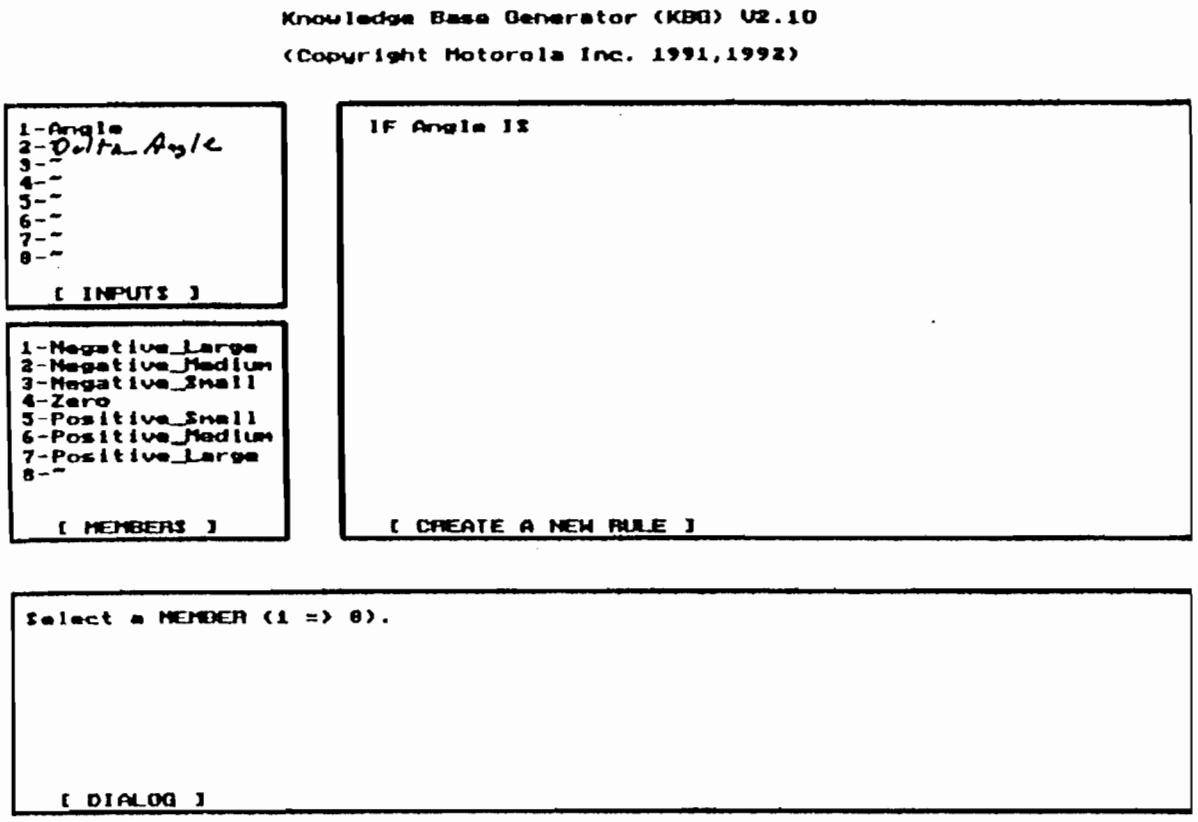
Enter the remaining input and output data in the same manner; then proceed to the heart of the matter, the rules. The rules and their treatment by the inference engine are the prime function of Fuzzy Logic. Rules are written by selecting inputs, outputs, and membership functions from a window. Use the **q** key to return to the Edit Menu. From this menu, the **r** key shows the following screen.



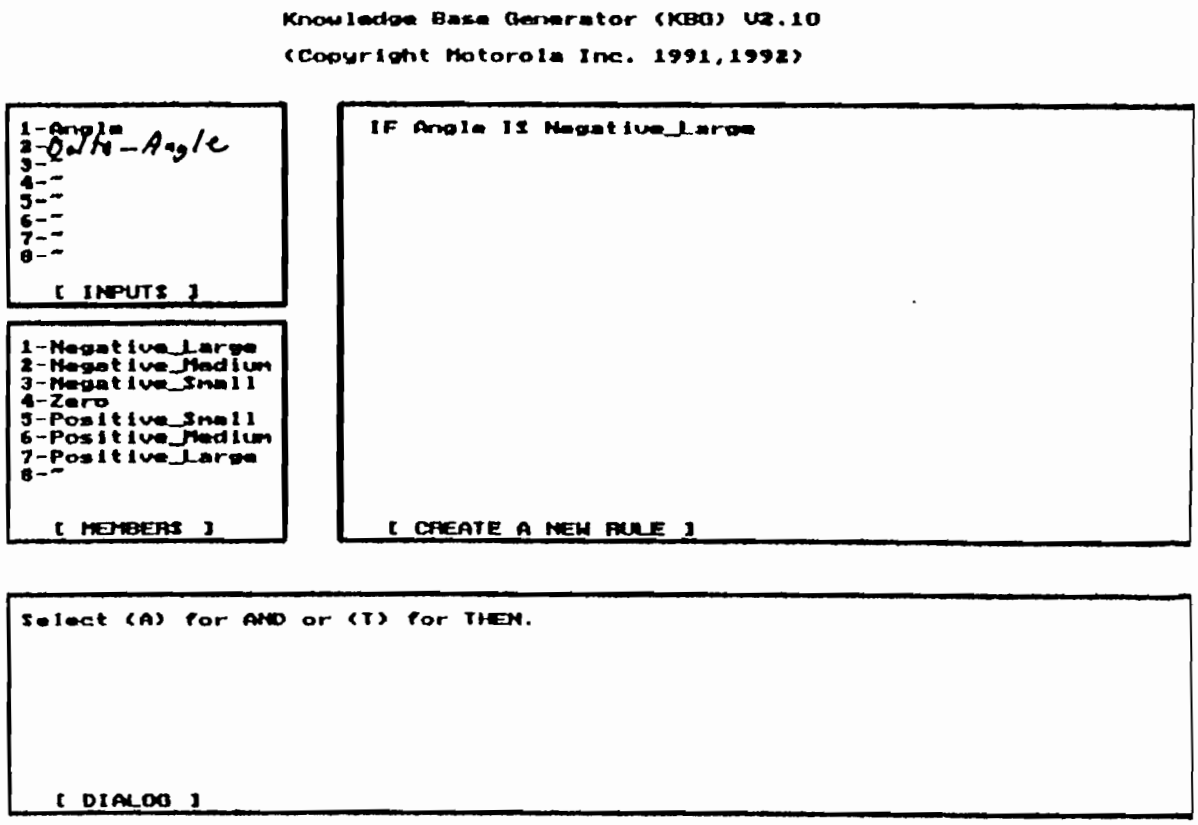
Select the **a** key to begin entry of the first rule. Build the rule in the right hand window, clause by clause. First, place the word **IF** in the new rule window. The next item in a rule is an input clause, which consists of an input and a membership function for that input. The program displays inputs in the left hand window and a prompt in the Dialog window to request an input number, as shown in the following figure.



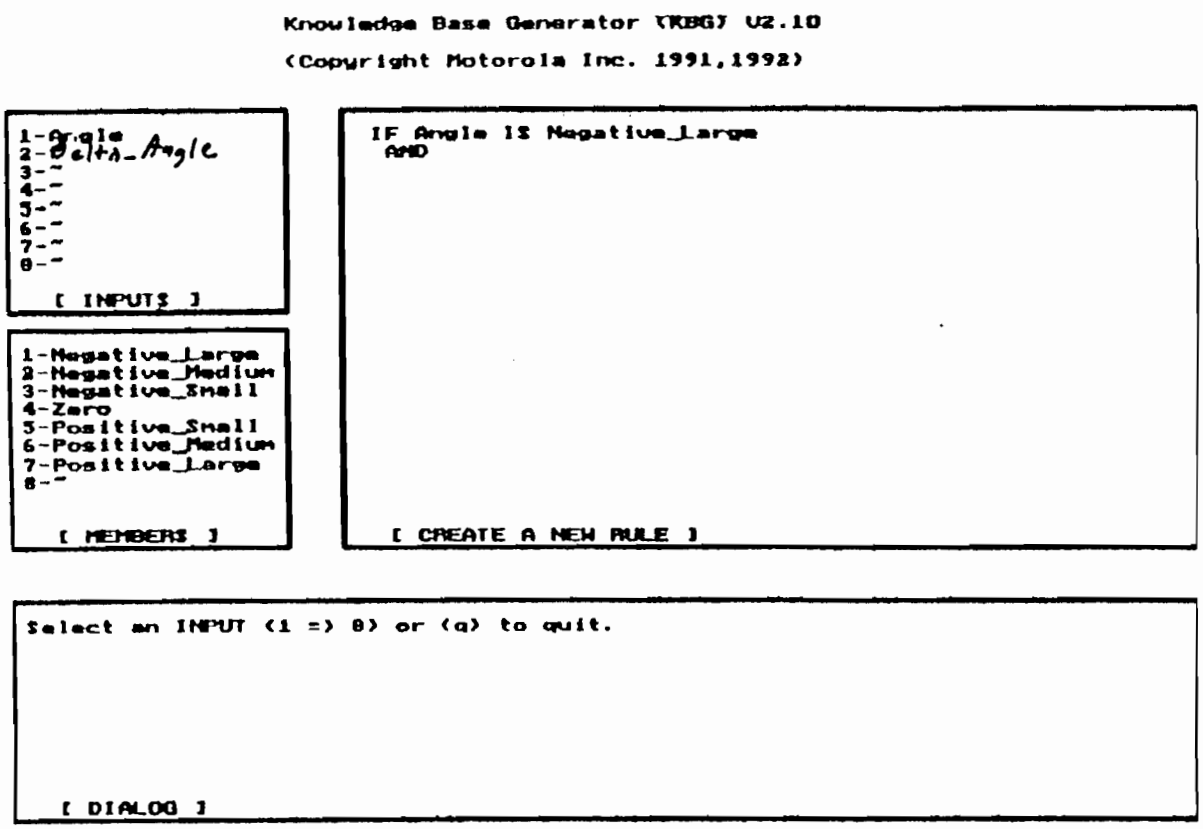
The first clause of the first rule states: *IF Angle IS Negative_Large*. Press the **1** key and the program displays the name Angle in the new rule window. The program displays a list of the membership functions for Angle in the Members window, as shown in the following figure.



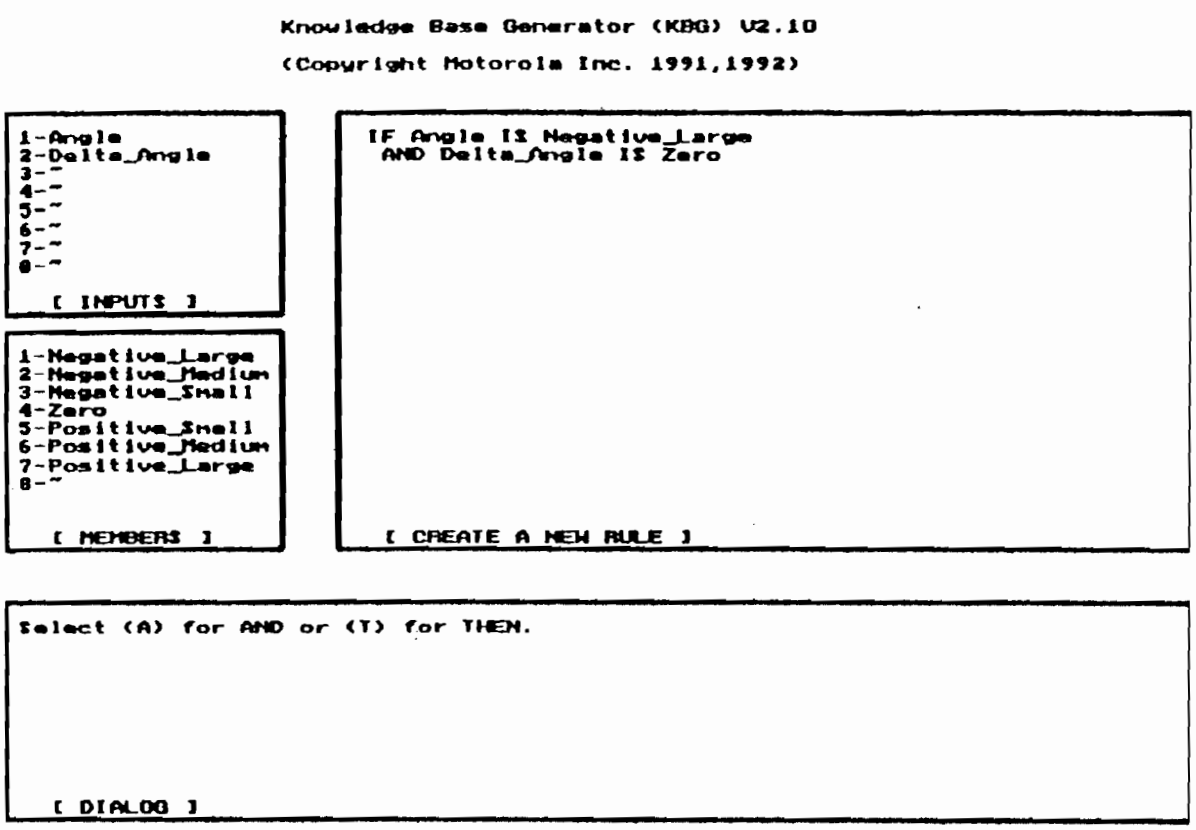
Now use the **1** key to select the membership function, **Negative_Large**. The program displays the member name in the new rule window and the dialog window asks for **AND** or **THEN**, as shown in the following figure.



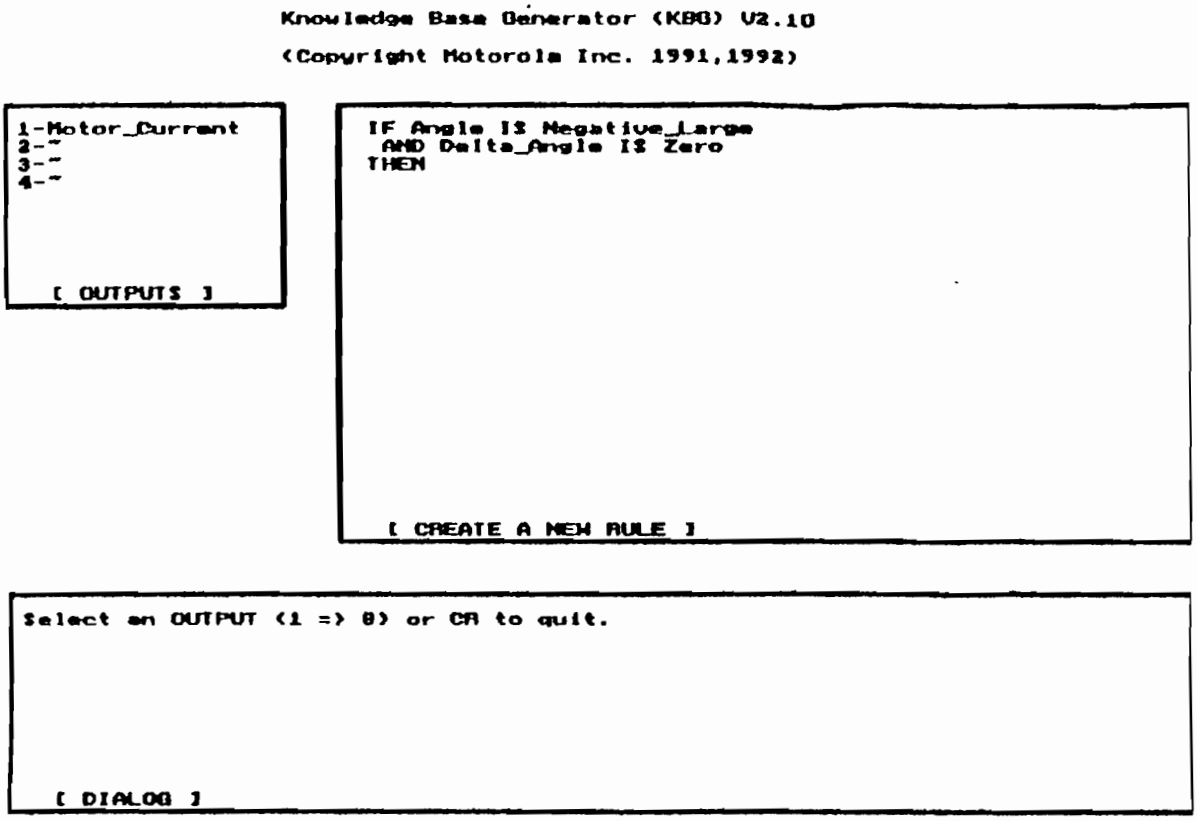
Rules that have one or more input clauses include the word **AND** to join the clauses. The rule that you are creating has two input clauses. Use the **a** key to add another input clause. The program retains the input windows on the screen, adds **AND** to the new rule window, and requests an input number in the dialog window. Note that each clause in the new rule window is written on a new line, with indention, to improve readability. The updated screen is shown in the following figure.



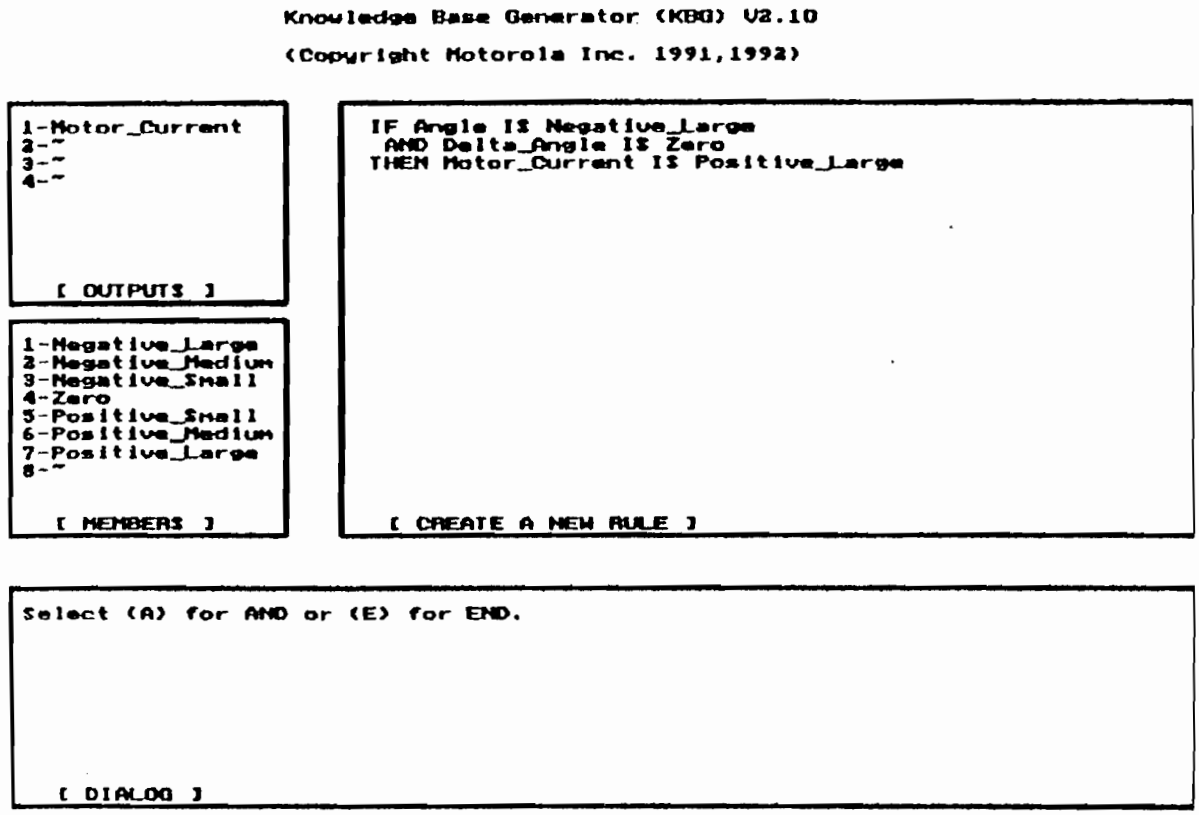
The second clause of the first rule states: *Delta_Angle IS Zero*. Press the **2** key to select **Delta_Angle** and press the **4** key to select **Zero**. The second and last input clause is complete. The dialog window offers the choice of another input clause or entry of the first output clause, as shown in the following figure.



Press the **t** key to enter an output clause. The program adds the word **THEN** in the new rules window, shows the output names in the right hand window, and requests an output selection in the dialog window, as shown in the following figure.



The first, and only, output clause of the desired rule states: *Motor_Current IS Positive_Large*. Press the **1** key to select **Motor_Current** and the function membership window appears, listing the membership functions for output #1. Press the **7** key to select **Positive_Large**, as shown in the following figure.



Since the rule you are creating has only one output clause, the rule is finished. Press the **e** key to terminate the rule. If the rule required more than one output clause, you could enter as many output clauses as desired by using the **a** key. The Dialog window goes away, and the completed rule is displayed as shown in the following figure.

```
Knowledge Base Generator (KBG) V2.10
(Copyright Motorola Inc. 1991,1992)

A - Add New Rule.
D - Delete a Rule.
Q - Quit.
[ COMMAND ]

1-IF Angle IS Negative_Large
     AND Delta_Angle IS Zero
   THEN Motor_Current IS Positive_Large

[ CURRENT RULES ]
```

This completes entry of one rule. Enter all the remaining rules in a similar manner. When rule entry is complete, use the **q** key to return to the Main Menu. First, save the knowledge base data in a disk file. Use the **s** key to select the save command. The save command opens a window that contains the names of currently existing knowledge base files. You could select one of these and write your new knowledge base to one of these. However, type the word **new** in the window. This creates a new file named **new.knb** and writes the new knowledge base data into it. An example screen showing rules 13 through 15 is shown in the following figure.

```
Knowledge Base Generator (KBG) V2.10
(Copyright Motorola Inc. 1991,1992)

A - Add New Rule.
D - Delete a Rule.
Q - Quit.
[ COMMAND ]

13-IF Angle IS Positive_Medium
      AND Delta_Angle IS Zero
    THEN Motor_Current IS Negative_Medium

14-IF Angle IS Positive_Large
      AND Delta_Angle IS Zero
    THEN Motor_Current IS Negative_Large

15-IF Angle IS Zero
      AND Delta_Angle IS Negative_Small
    THEN Motor_Current IS Positive_Small

[ CURRENT RULES ]
```

This file now contains all the knowledge base data that you have just entered. It is the source code for the knowledge base. However, this is not the data required by

the inference engine. The data must be scaled and encoded into constant bytes for the inference engine. Motorola assemblers require constant byte data to be coded as FCB directives. This is the last and most important function that KBG must perform. Use the **a** key to create an assembly language file, named **new.asm**.

This concludes the tutorial showing the use of KBG to create the Knowledge base. File **new.asm** must now be merged with the source code for the inference engine **fuzzy05b.asm** or **fuzzy11b.asm** to create a fuzzy logic processor. Other user code used to perform I/O, non-fuzzy data processing, and conversion of data to/from fuzzy logic may also be added to the file. The following DOS commands could be used to perform the task, assuming that an MC68HC11 is the target MCU:

```
C:> copy new.asm fuzz_app.asm

C:> type fuzzy11b.asm >>fuzz_app.asm

C:> type othercode.asm >>fuzz_app.asm
```

Now the knowledge base, inference engine, and other user code is in a single file. The following DOS command assembles the application. This command line assumes the use of the Freeware assembler, AS11.EXE, which generates a listing, **fuzz_app.lst**, and an S-record file, **fuzz_app.s19**.

```
C:> as11 -l fuzz_app.asm >fuzz_app.lst
```

You can program the file, **fuzz_app.s19**, into PROM or download it into a development system to be debugged.